

## UNIDAD 2: PROGRAMACIÓN A BAJO NIVEL (8/16 bits)

SET DE INSTRUCCIONES DE MOVIMIENTO DE INFORMACIÓN:  
(CPU ↔ CPU / CPU ↔ Memoria)



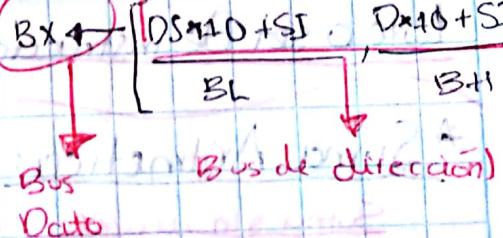
Ejemplo: \*MOV AX,BX      (CPU ↔ CPU)  
 AX → BX      16 bits

\*MOV AL,AH      (CPU ↔ CPU)  
 AL ← AH      8 bits

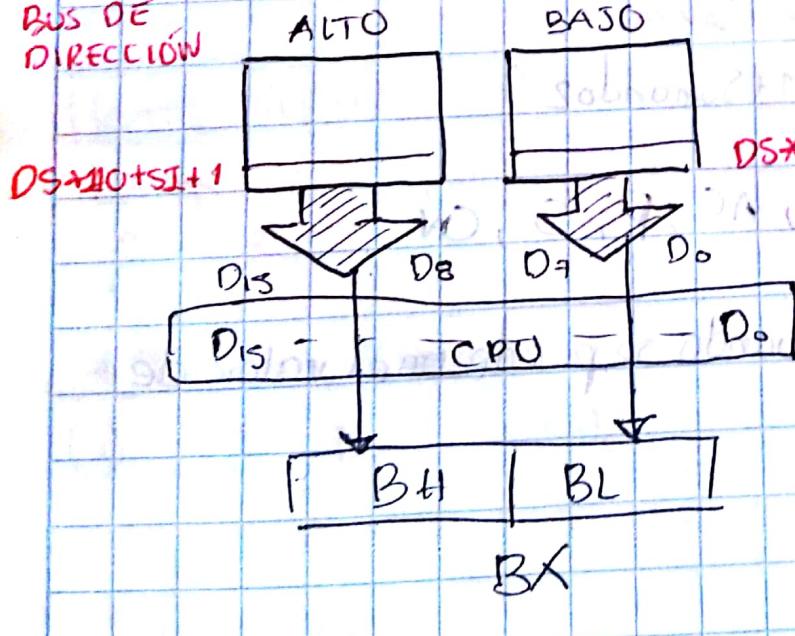
Bus de Control



MOV BX,[SI] → 16 bits



BUS DE DIRECCIÓN

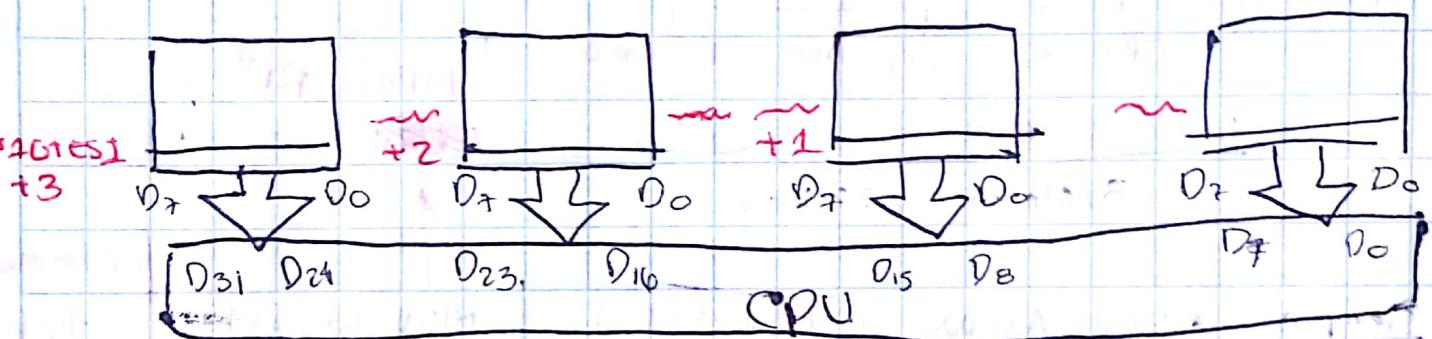


MOV EBX, [ESI]

Byte más bajo

$$EBX \leftarrow [DS * 10 + ESI + DS * 10 + ESI + 1 + DS * 10 + ESI + 2 \\ + DS * 10 + ESI + 3]$$

Byte más alto



Lote o set de instrucciones aritméticas (Suma y Resta).

### # Suma Aritmética:

Suma sin acarreo

① ADD sumando1, sumando2

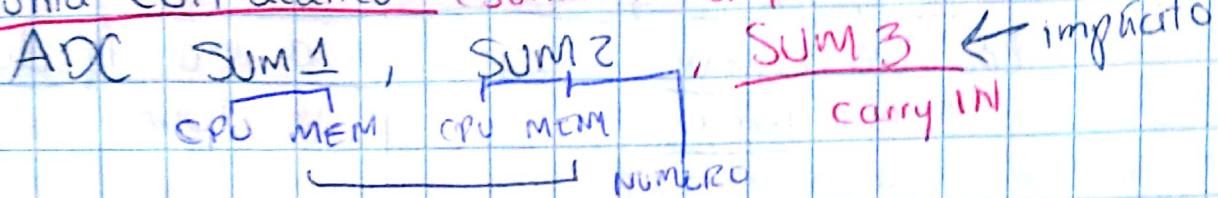
Desarrollo: CPU MEM CPU MMU NUMERO

$$\text{Sumando1} \leftarrow \text{Sumando1} + \text{Sumando2}$$

Banderas Afectas : Z, C, AC ; P, S, OF

Instrucciones Destructivas : Cuando se pierde el valor de uno de los sumandos

## (2) Suma CON acarreo (Sumador completo)

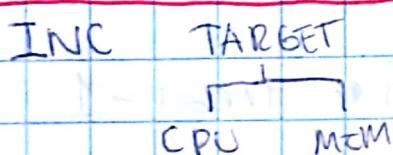


Desarrollo:      explícitos

$$\text{Sum } 1 \leftarrow \text{SUM}_1 + \text{SUM}_2 + \text{CF}$$

El carry siempre se suma en la columna menos significativa.

## (3) SUMA INCREMENTAL



Desarrollo:

$$\text{TARGET} \leftarrow \text{TARGET} + 1$$

$$\begin{array}{r} \text{TARGET} = \text{FF} \\ + 1 \\ \hline (1)00 \end{array}$$

↑ No lanza la bandera carry como el anterior quasi

FOR X=3 Step 1

$$W = R + S \leftarrow$$

Next X ↗

Si activara la flag afectaría las instrucciones dentro del for

## Resta ARITMÉTICA

## SUSTRACCIÓN

Los cpus intel planifican la resta en una analogía ~~de base 10~~ igual a la de base 10

## \* RESTA SIN PRESTAMO



Desarrollo:

$$\text{MIN} \leftarrow \text{MIN} - \text{SUSTRAENDO}$$

BANDERAS AFFECTAS: TODAS

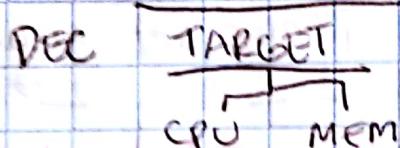
## Resta con préstamo

(2) SUB	$\frac{\text{MIN}}{\text{CPU MEM}}, \frac{\text{SUST}_1}{\text{CPU MEM}}, \frac{\text{SUST}_2}{\text{CF}}$	<u>Banderas Afectedas:</u> TODAS
---------	--	-------------------------------------

Desarrollo: MIN  $\leftarrow$  MIN - SUST<sub>1</sub> - SUST<sub>2</sub>

Como en el caso de la suma con carry, aquí el SUST<sub>2</sub> se resta en la columna menos significativa

## (3) DEC Resta Desremental



Desarrollo:

$$\text{TARGET} \leftarrow \text{TARGET} - 1$$

Banderas Afectedas: Todas

$$CX = 00_{16}$$

$$\text{DEC CX}$$

$$CX = FF$$

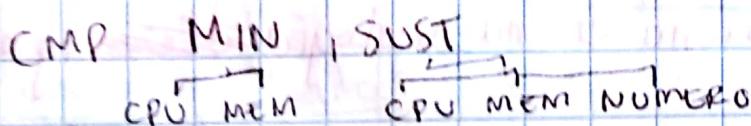
$$CX = FF$$

$$\text{DEC CX}$$

$$CX = FE$$

} es cíclico

## (4) RESTA NO DESTRUCTIVA "COMPARACION"

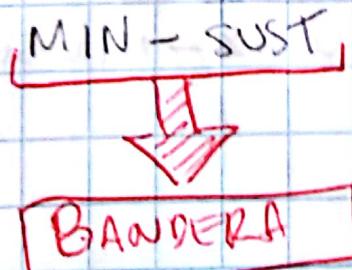


No hayalmacenamiento

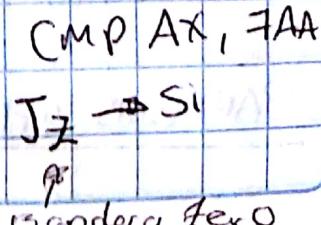
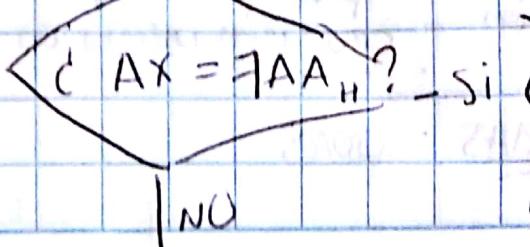
cuantitativo solo

qualitativo el status

DESARROLLO:



Banderas Afectedas: TODAS



CAPÍTULO 3  
 del libro  
 Examen 1 von Neumann (8 bits)  
 Modos de direccionamiento Bloques Pentium  
 / Registro CPU intel Complemento A2  
 / Memoria por segmentación

### Memonicos

## INSTRUCCIONES DE SALTO: (JUMP, BRANCH, SKIP)

→ SALTO NO CONDICIONAL (1)

→ SALTO CONDICIONAL (2)

1 y 2 ← RELATIVO

ABSOLUTO

CLASIFICACIÓN POR LA RAZÓN

EXISTENTE PARA SALTAR

CLASIFICACIÓN POR LA FORMA EN LA CUAL SE AFECTA AL "JP" O "PC" PARA UBICAR AL CPU A CAMBIOS DE NAVEGACIÓN

## INTEL Y SU PROPIA CLASIFICACIÓN DE SALTO

### INTER SEGMENTO

CUANDO SE PUEDE SALIR DEL SEGMENTO DE CÓDIGO EN CURSO.



### INTRA SEGMENTO

CUANDO NO PODEMOS SALIR DEL SEGMENTO DE CÓDIGO EN CURSO.

DEBIDO A QUE NO EXISTE ASIGNACIÓN NUMÉRICA DEL "CS" Y LA CICLICIDAD PROTEGE LAS FRONTERAS EN EL "JP"

### SALTOS LEJANOS

CUANDO SALTAMOS MÁS ALLÁ DEL SEGMENTO DE CÓDIGO EN CURSO, DISTANCIAS SUPERIORES A 16K PROPORCIONES

### SALTOS CERCANOS

NO DEJAMOS "CS" EN CURSO PERO LAS DISTANCIAS E (+16K, -16K)

### SALTOS CORTOS

NO DEJAMOS "CS" EN CURSO Y LAS DISTANCIAS E (+127, -127)

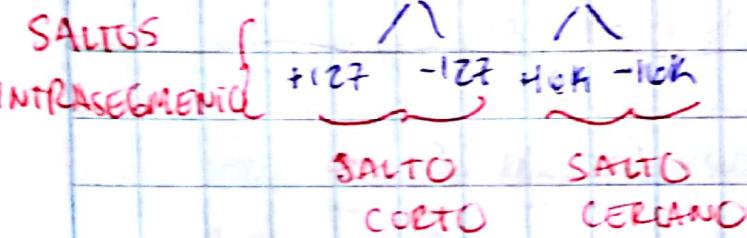
## SALTOS NO CONDICIONALES RELATIVOS

JMP  $\frac{e}{\text{offset}}$

DESPARDO:

JMP  $e$

5 bits 16 bits



$$IP \leftarrow IP + e$$

SOMA  
ARITMÉTICA

$e$  numero  
con  
SIGNO

## SALTOS CONDICIONALES RELATIVOS

J FLAG

$\frac{e}{\text{offset}}$

SALTO INTRA SEGMENTO

SALTO CORTO O SALTO LEJANO  
DEPENDE TAMAÑO  $e^n$

SIMPLÉS

I  
H2

F NC  
C DE  
P0

B = below

A = Above

E = equal

G = Greater

L = Less

E = equal

COMPLEJAS

SIN SIGNO

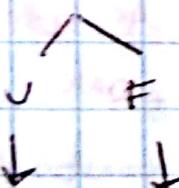
A	B
AE	BE
E	NE

CON SIGNO

G	L
GE	LE
E	NE

DESPARDO:

① EVALUA FLAG



$IP \leftarrow IP + e$        $IP \leftarrow IP + 1$

en el resultado el sum salta de celda en celda

II Absoluto II salta directamente al especificado

SALTOS NO CONDICIONADOS ABSOLUTOS

JMP  
registro  
16 bits

JMP [registro]  
16 bits

DESARROLLO:

IP<sub>A</sub> → registro

IP<sub>A</sub> → [DS\*10  
+ registro]

DS\*10 + 1 + Regis:

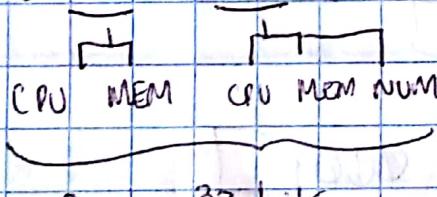
NOTA: Las operaciones lógicas  
se desarrollan por  
"bit wise"

INSTRUCCIONES LÓGICAS

(1) MULTIPLICACIÓN LÓGICA

(1) DESTRUCTIVA

AND OP<sub>1</sub>, OP<sub>2</sub>



TEST OP<sub>1</sub>, OP<sub>2</sub>

Desarrollo:

OP<sub>1</sub> ∨ OP<sub>2</sub>

qualitative

↓

FLAG

Desarrollo:

OP<sub>1</sub> → OP<sub>1</sub> ∨ OP<sub>2</sub>

QUALITATIVO

↓

FLAG

EJEMPLO:

AND AX, BX

AX : 1111 0101 0001 1100

BX : 0000 1111 1111 0001

AX : 0000 0101 0001 0000

SPZ 0 C

\*\*\* 0 D

FFF 0 P

00000 0 0

## APLICACIÓN DE LA "AND" AL FILTRO DIGITAL

PATRÓN DIGITAL : XXXX XXXX AND ALT, 10000000  
 MÁSCARA : 00000 1111 ^  
00000 XXXX

## (2) SUMA LÓGICA

### ① DESTRUCTIVA

DESEARROLLO :

OR OP1, OP2  
 ↗ ↗  
 CPU MEM CPU mem num

OP1 ← OP1 + OP2  
 ~  
 Cuantitativo  
 ↓  
 FLAG

EJEMPLO: OR, DX, FFC00H

$DX = 1001\ 1111\ 00000\ 0001$   
 $FFC0 = 1111\ 1111\ 00000\ 0000$   
 $DX = \underline{1001}\ 1111\ 00000\ 0001$

## APLICACIÓN DE LA "OR" AL FILTRO DIGITAL

PATRÓN DIGITAL : XXXX XXXX  
 MÁSCARA : 0000 1111 ^  
XXX 1111

MEM = Registros

CPN = Direccionamientos

OR EXCLUSIVO

XOR OP<sub>1</sub>, OP<sub>2</sub>

Diagram illustrating a 32-bit memory address bus structure:

- CPU MEM**: 8 bits
- CPU MEM NUMBER**: 8 bits
- REG**: 4 bits
- M.D.O index**: 4 bits

The total width of the bus is 32 bits.

DESARROLLO:  $OPI_1 \leftarrow OPI_1 \oplus OPI_2$

BAD, AFCC : OV S. P AC C Z  
O X X ? O \*

## Aplicación filtro Digital.

Patrón desconocido =  $\times \times \times \times$

$$\text{MÁSCARA} = \frac{\phi\phi\phi\phi}{xxx} \quad \overline{1111}$$

$y$	$x$	$z$
0	0	0
0	1	1
1	0	1
1	1	0

## Complemento A UNO:

NOT TARGET

EJ: NOT AH

$$A \oplus = \begin{matrix} 1 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{matrix} \text{ NOT}$$

$$AH = 0110\ 0001$$

Desarrollo: TARGET 4 — TARGET

No afecta ninguna bandera

## Complemento A DOS:

```

    graph LR
      NEG[NEG] --> CPU[CPU]
      TARGET[TARGET] --> CPU
      CPU --> MEM[MEM]
  
```

TARGET → TARGET + 1

OV SP AC C Z  
\* \* \* \* \*

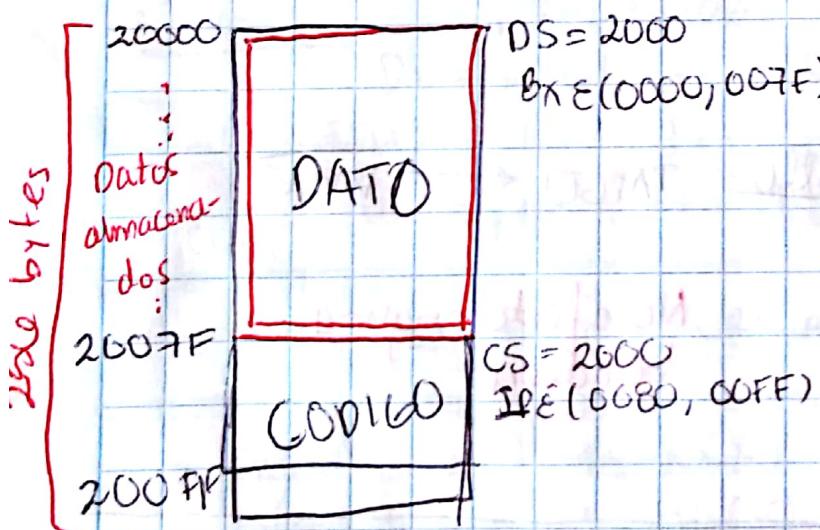
$\text{XOR AX, AX} = \text{MOV AX, 0000}$  (Limpia)

cod instru = cod operación | cod instru  
 longitud = 2 bytes

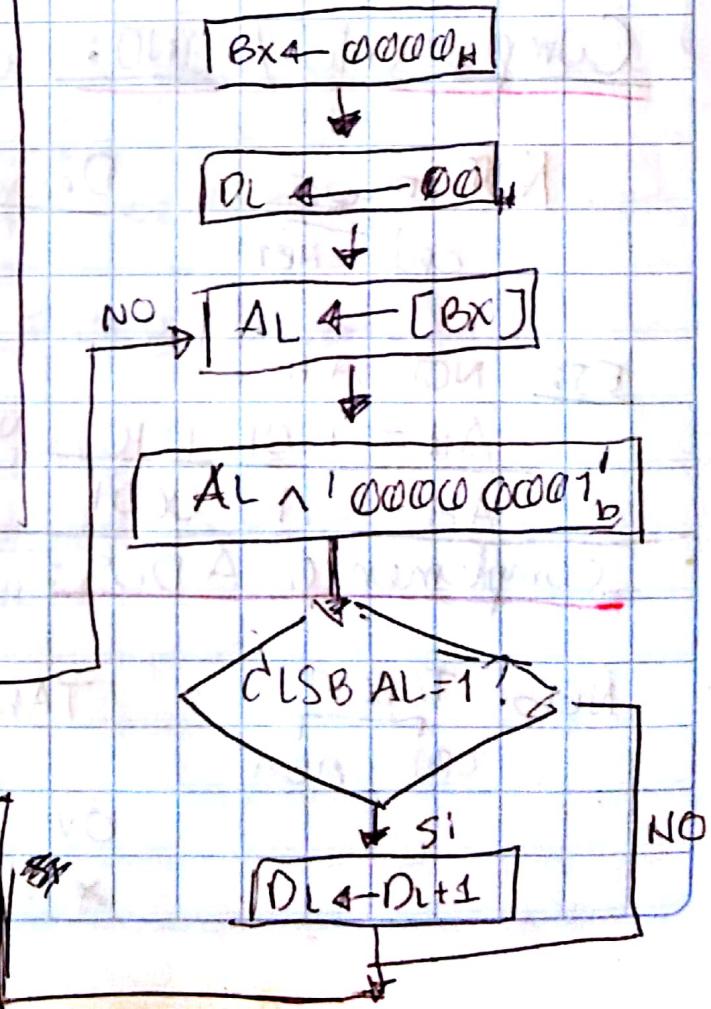
Suponga que posee un computador armado con un CPU intel, software para emular programas a bajo nivel y memoria RAM libre de la posición 20000 → 200FF. Suponga que en la posición 20,000 a la 2007F han sido previamente almacenados datos tamaño byte.

Proceda a diseñar un software que detecte cuantos de estos bytes son impares, y almacene el numero de ellos en el registro bajo de 8 bits del registro DX. Al finalizar la tarea el programa finalizará también.

### ① División Memoria:



### ② Diagrama de flujo:



## PROGRAMA A BAJO NIVEL

### EQUIGAN

### DOS MEM (COPICO)

### NOMÍNICO

### EQU SALIDA

### COMENTARIO

2000:0080

XOR BX,BX

Inicializo puntero  
index de memoria

2000:0082

MOV DL,0AH

Inicializo contador  
de impares

[NC] → 2000:0084

MOV AL,[BX]

transf. dato/byte a CPU

TEST AL,00000010

AL=max xxxx  
[AL=0000 0001

JZ → [F:TM]

0000 000X

JNC DL,00000010

Investigo el siguiente

JNC BX

byte

CMP BX,0080H

JNZ →

[F:NC]

FIN

Windows 16 bits

→ debug

→ -r (Muestra valor de los registros)

→ a

0B05:0100 MOV AX,00002

0B05:0103 MOV BX,10FF

^C (Break)

-G 108 → compila todo hasta 108

MOV AX, m<sub>2</sub> m<sub>1</sub>

Cod Inst = B8 m<sub>1</sub> m<sub>2</sub>

↑  
cod operación

ADD AX,13X

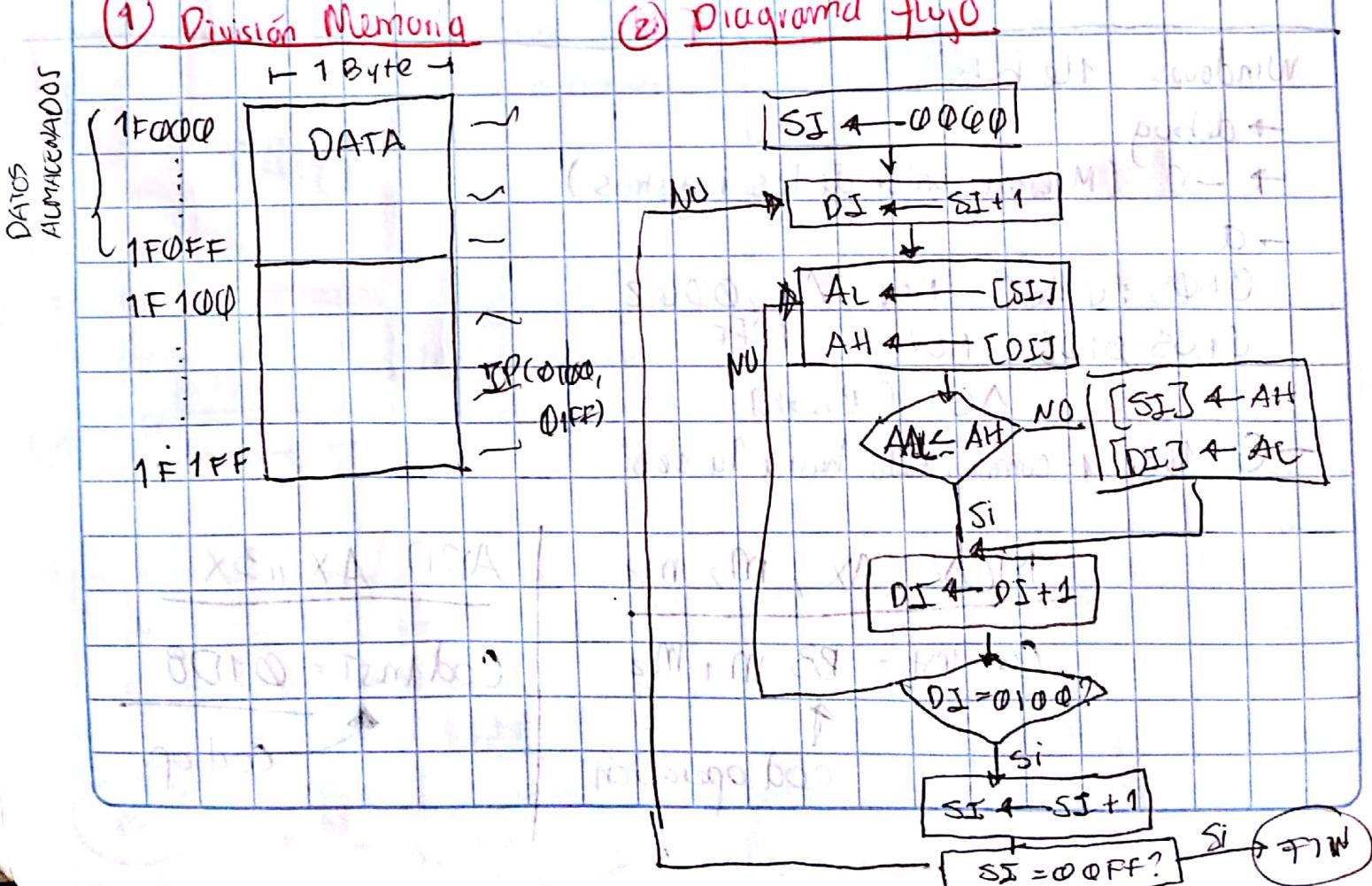
Cod Inst = 01D8

↑  
cod op.

~~NOP~~ → No hacen nada? Sirve para dejar ese espacio reservado  
 → Trace ~~11111111~~ va sumando 1 al IP para ejecutar linea por linea

Un computador armado con un CPU intel con software para emular un programa a bajo nivel y RAM disponible de la posición 50,000 a la posición 50,FFF. Supongamos que existen datos tamaño byte, pre-escritos de la posición 50,000 a la posición 50,OFF. Diseñe un programa a bajo nivel por medio del cual ordene los datos allí encontrados en forma ascendente (Dejando el más chico en la 50,000). Al terminar la tarea el programa finalizará también.

Ejemplo: DISEÑAR UN PROGRAMA A BAJO NIVEL QUE SEA CAPAZ DE ORDENAR LOS DATOS EN MEMORIA  
 (LO MISMO DE ANTES PERO DE 1F0000 → 1F0FF) → 1F100 → 1F1FF



### (3) PROGRAMA A BAJO NIVEL

ETQ LLEGAN

POS MEMORIA

NEMONICO

ETQ SALEN

COMENTARIO

1FOQ: 0100

XOR SI, SI

MOV DI, SI

INC DI

MOV AL, [SI]

MOV AH, [DI]

Cmp AL, AH

JBE

[HOY]

MOV[SI], AH

MOV[DI], AL

INC DI

Cmp DI, 0100

JNE

[OLP]

INC SI

Cmp SI, 00 FF

JNE

[OJN]

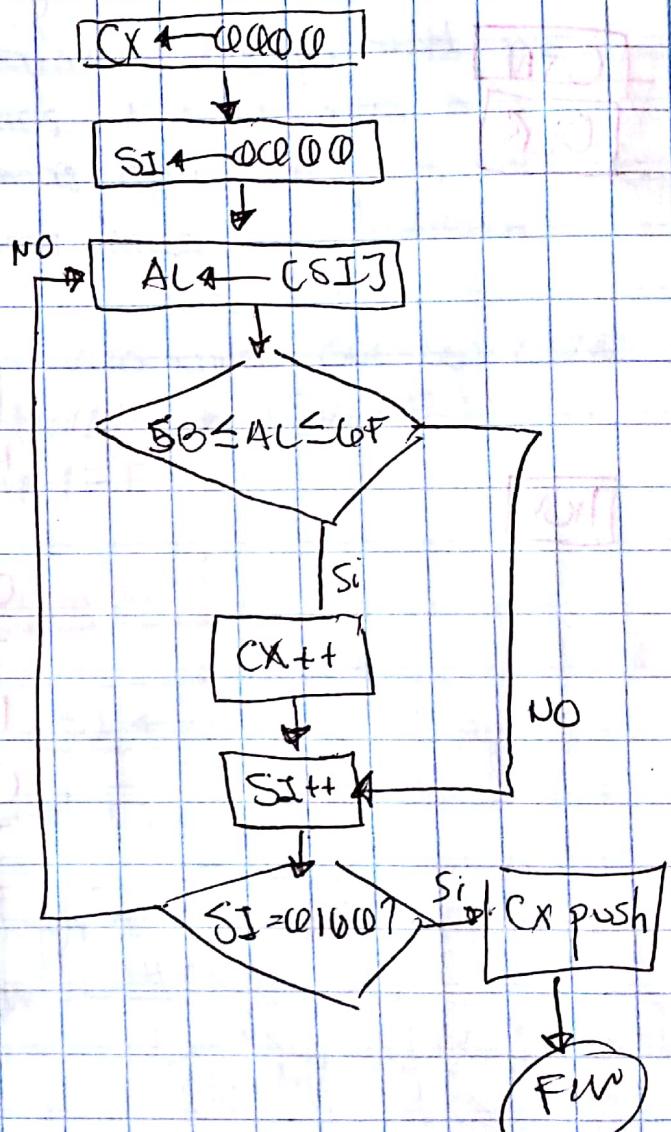
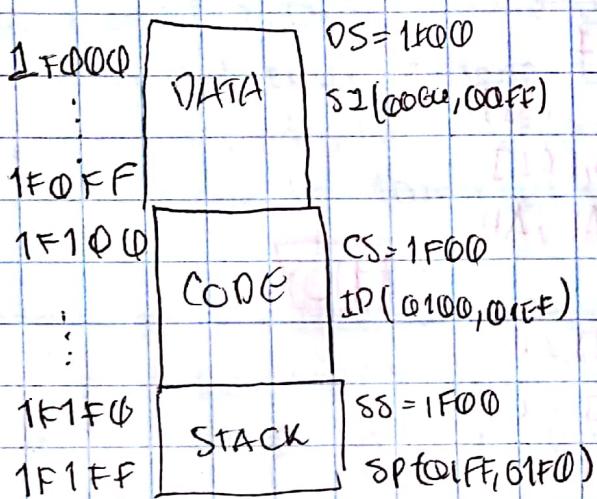
FIN

PROGRAMA QUE SEA CAPAZ DE ENCONTRAR CUANTOS DATOS

TAMAÑO BYTE SE ENCUENTRAN EN EL RANGO DE  $5B_4 \rightarrow 6F_4$ .

DE LOS PRE-ESCRITOS EN EL RANGO DE LA  $1F0000 \rightarrow 1FOFF$ .

DESE EL NUMERO DE HUAZGOS EN STACK. RAM LIBRE DE LA  
 $1F1000 \rightarrow 1F1FF$



XOR CX, CX

XOR SI, SI

MOV AL, [SI]

CMP AL, 5B

JB [FUR]

CMP AL, 6F

JA [FUR]

INC CX

INC SI

CMP SI, 010001

JNE [VDE]

PUSH CX

FIN

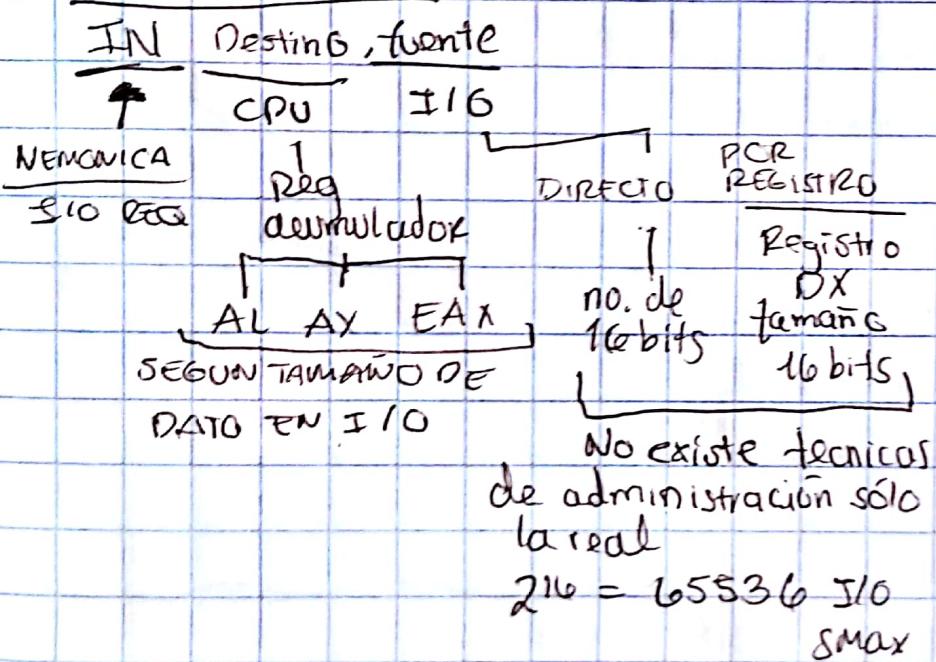
VDE

FUR

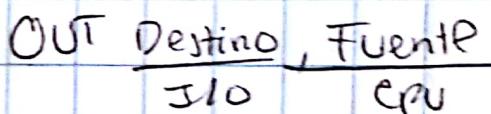
## INSTRUCCIONES DE MOVIMIENTO DE INFORMACION

## DESDE Y HACIA PUERTOS

## LECTURA PUERTOS



## ESCRITURA PUERTOS



Ejemplo: Diseñar un programa a bajo nivel, que sea

## INSTRUCCIONES DE DESPLAZAMIENTO

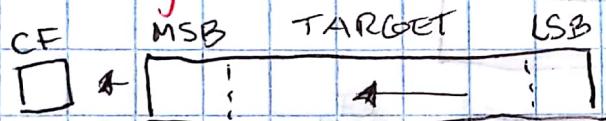
(SHIFT)

→ Desplaz Aritmético  
→ II

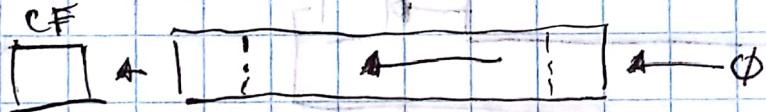
→ Desplaz  
→ II

Izquierda  
Derecha

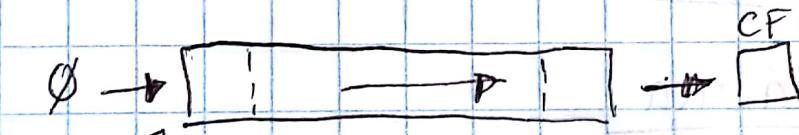
SHL



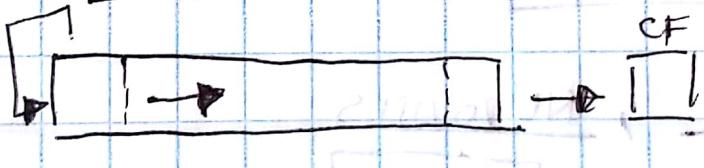
SAL



SHR



SAR



movimiento lógico a la izquierda

movimiento numerico a la izquierda

movimiento lógico a la derecha

movimiento aritmético a la derecha

## ESTRUCTURA

## SINTÁCTICA

NÉMÓNICO      TARGET      REGISTROS

CPU  
1  
DIRECTO

, NO SHIFT'S  
REGISTRO CPU  
CL

{ Ej: Multiplicar AX - 10

SHL AX, 1 ;  $AX = AX \times 2$

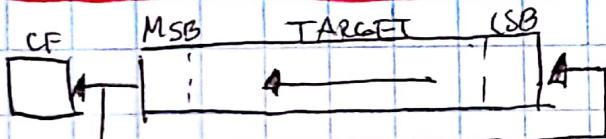
MUL BX, AX ;  $BX = AX \times 2$

SHL AX, 2 ;  $AX = AX \times 2$

ADD AX, BX ;  $AX = AX + BX$   
 $= AX - 10$

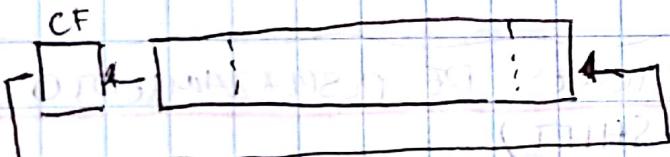
## INSTRUCCIONES DE ROTACIÓN : (ROTATE)

ROL



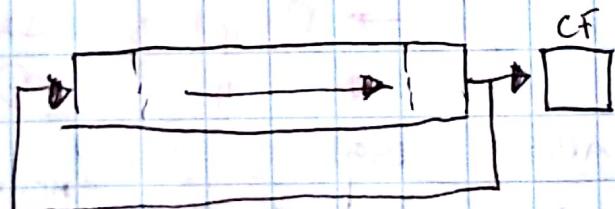
ROTACIÓN A LA IZQUIERDA  
SIN CARRY

RCL



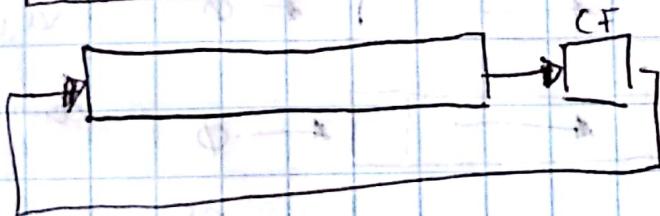
ROTACIÓN A LA IZQUIERDA CON  
CARRY

ROR



ROTACIÓN A LA DERECHA SIN  
CARRY

RCR



ROTACIÓN A LA DERECHA CON  
CARRY

## ESTRUCTURA SINTÁCTICA

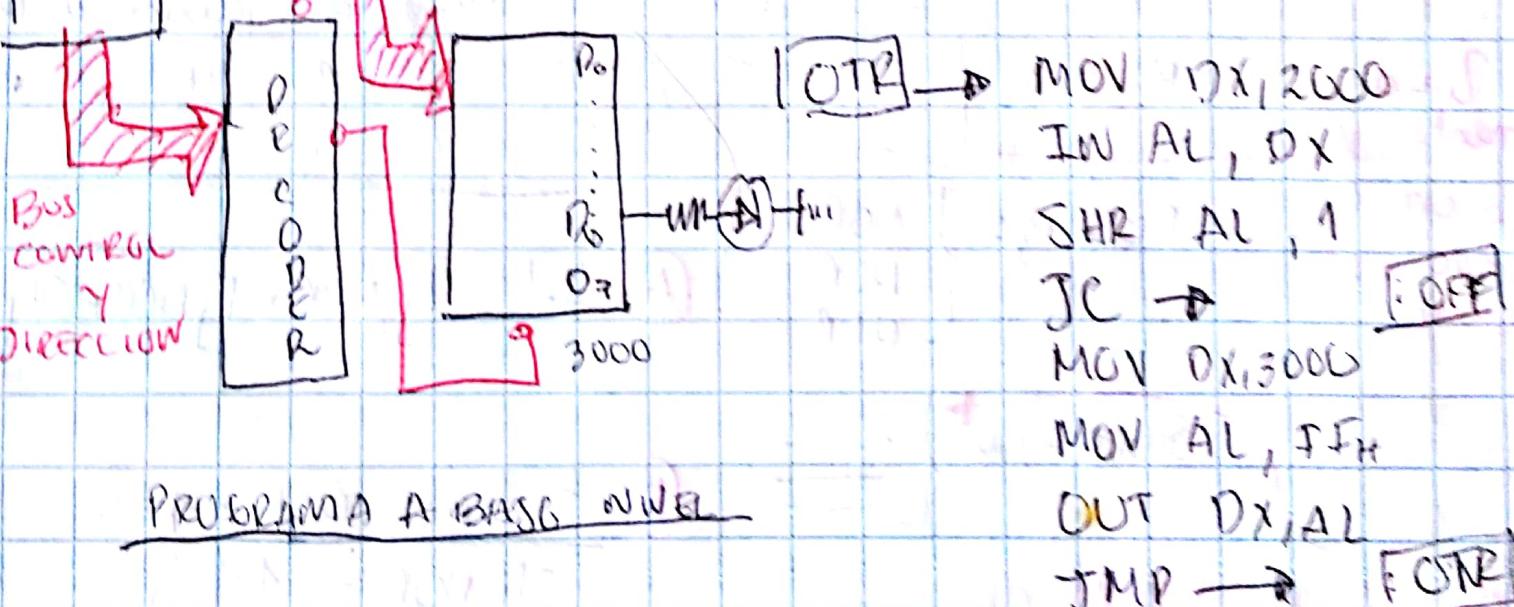
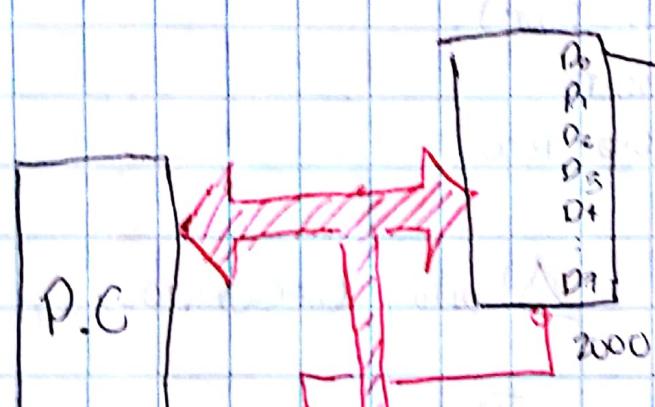
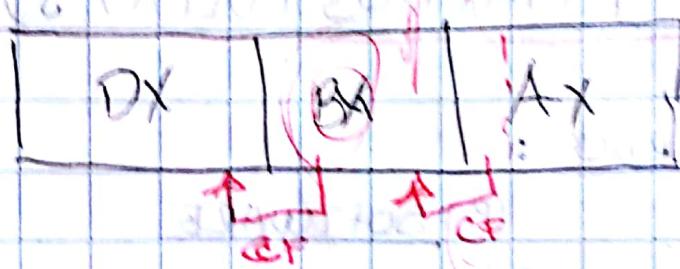
NÉMÓNICO , TARGET , NO ROTATES

CPU  
1  
REGISTROS

NUMERO CL

## EJEMPLO: UN SÓLO NÚMERO

SHL AX, 1  
RCL BX, 1  
RCL DX, 1



EMULAR RECONTER DE 13x9